JOURNAL OF COMPUTER AND SYSTEM SCIENCES 29, 183–197 (1984)

Time–Space Tradeoffs for Matrix Multiplication and the Discrete Fourier Transform on Any General Sequential Random-Access Computer

YAACOV YESHA

Department of Computer Science, University of Toronto, Toronto, Canada M5S 1A7

Received September 9, 1982; revised February 28, 1984

On any general sequential model of computation with random-access input (e.g., a logarithmic cost RAM or a Turing machine with random-access input heads) the product *time* · *space* is:

(1) Not $o(N^2)$, hence not $o((n/\log n)^2)$, for computing the discrete Fourier transform over finite prime fields, even when each entry in the input vector has length $O(\log N)$. Here N denotes the number of entries and n denotes the input length.

(2) $\Omega(M^3)$, hence not $o((n/\log n)^{1.5})$ for M by M matrix multiplication over the integers or over finite prime fields, even when each entry in the matrices has length $O(\log M)$.

For this range of entries length these lower bounds on *time* · *space* coincide, up to a $\log n^{O(1)}$ factor, with the upper bounds achieved by the straightforward arithmetic algorithms. Time–space tradeoffs for the discrete Fourier transform and for matrix multiplication on the restricted model of a straight-line algorithm were previously obtained by Grigoryev ("Notes on Scientific Seminars 60," pp. 38–48, Steklov Math. Inst., Leningrad, 1976. [Russian]) Ja'Ja' ("Proceedings 12th Annual ACM Sympos. Theory Comput., 1980," pp. 339–349) and Tompa (University of Toronto, Dept. of Comput. Sci. Tech. Report 122/78). The model considered is *general*, meaning that it is *not* restricted to performing a sequence of arithmetic operations. Arbitrary bit processing and branching is allowed.

1. INTRODUCTION

The results of this paper apply to any reasonable model of computation which is *general*, *sequential*, and with *random-access input*.

General means that the model can access any bit of the input, and the computation does not have to be of a form restricted by the mathematical structure of the problem or by not allowing branching according to intermediate results. In *structured* models, however, the computation is of a restricted form. For example: comparison trees for sorting, arithmetic straight-line programs for arithmetic computations. Boolean straight-line programs (see, e.g., [7, 12]) are not general. This is since in a straight-line program the order of performing the operations cannot depend on the values of intermediate results. [2] is an excellent source for a comparison between general and structured models. See also [3, 12].

0022-0000/84 $3.00

Sequential, as opposed to *parallel*, means that the number of active components is constant and does not depend on the size of the input.

Random-access input means that in one unit of time the reading head of our computer can transfer to any bit of the input. A logarithmic cost RAM (see, e.g., [1]), and a Turing machine with random-access input heads (see, e.g., [3, 12]) are included in our models.

In their pioneering paper [3], Borodin and Cook prove that on any general sequential model of a random-access computer, sorting of N numbers in the range $[1, N^2]$ requires *time* $\cdot$ *space* $= \Omega(N^2/\log N)$. This result has recently been improved by Reisch and Schnitger to $\Omega(N^2 \operatorname{loglog} N/\log N)$ [11]. Time–space tradeoff results for natural problems preceding [3] rely on restricted models. [5 and 6] assume *tape input*, which means that in one unit of time a reading head can move only one position to the left or right. Thus there is no random-access input. [12, 4, 10, and 7] deal with nongeneral models.

In this work we apply the techniques developed in [3] to algebraic computational problems. We obtain the following result:

THEOREM 1.1. *On any general sequential model of computation with random-access input, the product time* $\cdot$ *space is*:

(1) *Not* $o(N^2)$, *hence not* $o((n/\log n)^2)$, *for computing the discrete Fourier transform over finite prime fields, even when each entry in the input vector has length* $O(\log N)$. *Here* N *is the number of entries in the input vector, and* n *is the input length.*

(2) $\Omega(M^3)$, *hence not* $o((n/\log n)^{1.5})$, *for* M *by* M *matrix multiplication over finite prime fields or over the integers, even when each entry in the matrices has length* $O(\log M)$.

Here *time* (resp. *space*) denotes the maximum number of steps (resp. amount of storage space) used by the program, when the maximum is taken over all inputs with the same N (in (1)) or M (in (2)).

It should be emphasized that a *general* model is not required to perform, say, integer matrix multiplication only by performing arithmetic operations on the entries of the input matrices or on intermediate results obtained by such operations. Arbitrary processing of the input bits is permitted. Our results apply to such powerful models. While arithmetic operations seem natural in the context of matrix multiplication and the discrete Fourier transform, it could still be the case that faster algorithms may be obtained on general models. Hence indeed lower bounds for such models are of much interest (see also discussion in [3]).

We note that for the range of entries length mentioned above our lower bounds coincide, up to a $\log n^{O(1)}$ factor, with the upper bounds achieved by the straightforward arithmetic algorithms. We also note that no one has yet been able to prove that M by M integer matrix multiplication with entries of length $O(\log M)$ cannot be performed on an unrestricted model in, say, $O(n \log n^{O(1)})$ *time*. It is not

even known whether a time bound $O(n)$ is achievable or not. It follows from our result that no program for this problem operating in $O(\log n^{O(1)})$ *space* can also operate in $O(n \log n^{O(1)})$ *time*.

In order to obtain the above tradeoff results, we introduce the concept of (m, α)*mixing functions*. This is a property of functions which enables using Borodin and Cook's technique. We also introduce the *combination principle* (Lemma 3.1) which states that the (m, α)-mixing property is preserved under the combination of functions with disjoint sets of variables. This principle is very useful.

On the restricted model of an arithmetic straight-line program, Tompa [12] has obtained a time–space tradeoff for the discrete Fourier transform and Ja'Ja' [10] has obtained time–space tradeoffs for matrix multiplication and related problems. On the restricted model of a Boolean straight-line program, Grigoryev [7] has obtained a time–space tradeoff for mod 2 matrix multiplication and polynomial multiplication.

2. Preliminaries and Notation

In order to obtain the time–space tradeoff results we use a technique due to Borodin and Cook [3]. We formalize our version of their technique in Theorem 2.1, and we call it "Borodin and Cook's counting principle." We are going to state it in a form which suits our purposes, and not in the strongest form possible.

For standard definitions from computational complexity and in particular the resources *time* and *space*, see [1 and 8]. The computational problems considered in this work involve computing functions of a variable number of variables. The number of variables will depend on a parameter which we will usually denote by N. For example, for matrix multiplication N will denote the number of entries in each matrix.

We now introduce the following notation: p_l $(l = 1, 2,...)$ will denote the lth prime. For any prime I, $GF(I)$ will denote the finite field with I elements. For any natural number I, $Q(I)$ will denote the set of integers of absolute value at most I. For any ring D, an $I \times J$ matrix $\bar{z}$ whose *rows* are $\bar{z}_1, \bar{z}_2,..., \bar{z}_I$, where $\bar{z}_i = (z_{i1}, z_{i2},..., z_{iJ}) \in D^J$ $(1 \leqslant i \leqslant I)$ will be denoted by $(\bar{z}_1, \bar{z}_2,..., \bar{z}_I)^T$. An $I \times J$ matrix $\bar{x}$ whose *columns* are $\bar{x}_1, \bar{x}_2,..., \bar{x}_J$, where $\bar{x}_j = (x_{j1}, x_{j2},..., x_{jI})$ will be denoted by $(\bar{x}_1, \bar{x}_2,..., \bar{x}_J)$. The product matrix obtained by multiplying $\bar{z}$ by $\bar{x}$ will be denoted by $\bar{z} * \bar{x}$. The product of a matrix $\bar{z}$ by a vector $\bar{x}$ will be denoted by $\bar{z} \cdot \bar{x}$.

We are going to consider vectors over $GF(I)$ or over $Q(I)$, which are inputs and outputs to the programs which compute certain functions. In general, an element of a domain D will be represented in binary, using exactly $\lceil \log_2 |D| \rceil$ bits. A vector $(a_1, a_2,..., a_I) \in D^I$ will be represented by $\#a_1\#a_2\# \cdots \#a_I\#$.

Our computing device has the input written on its read-only input medium. If the output to be computed is $(y_1, y_2,..., y_J) = (b_1, b_2,..., b_J)$, then during the computation, pairs of the form (i, b_i) are printed out for $1 \leqslant i \leqslant J$. We do not assume any particular order of printing these pairs.

A field is called *prime* if it has no nontrivial subfields. A finite field is prime if and only if its number of elements is prime.

We now describe the functions for which we will show time–space tradeoffs.

(1) *DFT*: Discrte Fourier transform over finite prime fields.

Input: A prime p; $\bar{x} = (x_1, x_2, ..., x_N) \in D^N$, where $D = GF(p)$; w—a primitive Nth root of unity in D (it is assumed that N is such that w exists).

Output: $\bar{y} = (y_1, y_2, ..., y_N) = \bar{z} \cdot \bar{x} \pmod{p}$, where $\bar{z}$ is the matrix $\bar{z} = (\bar{z}_1, \bar{z}_2, ..., \bar{z}_N)^T$ such that $z_{ij} = w^{(i-1)(j-1)} \pmod{p}$.

(2) *MF*: Matrix multiplication over finite prime fields.

Input: A prime p; a matrix $\bar{z} = (\bar{z}_1, \bar{z}_2, ..., \bar{z}_M)^T \in D^N$, a matrix $\bar{x} = (\bar{x}_1, \bar{x}_2, ..., \bar{x}_M) \in D^N$, where $D = GF(p)$, $N = M^2$.

Output: A matrix $\bar{y} = (\bar{y}_1, \bar{y}_2, ..., \bar{y}_M) \in D^N$, where $\bar{y} = \bar{z} * \bar{x} \pmod{p}$.

(3) *MI*: Matrix multiplication over the integers.

Input: A natural number I; a matrix $\bar{z} = (\bar{z}_1, \bar{z}_2, ..., \bar{z}_M)^T \in D^N$, a matrix $\bar{x} = (\bar{x}_1, \bar{x}_2, ..., \bar{x}_M) \in D^N$, where $D = Q(I)$, $N = M^2$.

Output: A matrix $\bar{y} = (\bar{y}_1, \bar{y}_2, ..., \bar{y}_M) \in E^N$, $\bar{y} = \bar{z} * \bar{x}$, where $E = Q(I^2 M)$.

We now introduce, for each of *DFT*, *MF*, and *MI* a corresponding infinite family of functions. It will be clear that any program which computes one of the above functions can also compute the corresponding infinite family of functions. The introduction of infinite families of functions will enable us to use the concept of *computation graphs* [3].

(1) $F = \{F_l \mid l = 1, 2, ...\}$, corresponding to the special case of *DFT* in which $N = p - 1$. Let $D_l = GF(p_l)$, $N_l = p_l - 1$, w_l a fixed primitive N_lth root of unity modulo p_l.

$F_l: D_l^{N_l} \to D_l^{N_l}$ is given by:

$$F_l(x_1, x_2, ..., x_{N_l}) = (y_1, y_2, ..., y_{N_l}),$$

where $y_j = \sum_{i=1}^{N_l} x_i \cdot w_l^{(i-1)(j-1)} \pmod{p_l}$ $(1 \leqslant j \leqslant N_l)$.

(2) $MA = \{MA_l \mid l = 1, 2, ...\}$, corresponding to the special case of *MF* in which $M = p - 1$. Let $D_l = GF(p_l)$, $M_l = p_l - 1$, $N_l = M_l^2$, w_l a fixed primitive N_lth root of unity modulo p_l. Let $\bar{z}(l)$ be the fixed matrix $\bar{z}(l) = (\bar{z}(l)_1, \bar{z}(l)_2, ..., \bar{z}(l)_{M_l})^T$, where $z(l)_{ij} = w_l^{(i-1)(j-1)} \pmod{p_l}$ $(1 \leqslant i, j \leqslant M_l)$, then $MA_l: D_l^{N_l} \to D_l^{N_l}$ is given by:

$$MA_l(\bar{x}) = \bar{y} = \bar{z} * \bar{x} \pmod{p_l},$$

where $\bar{x} = (\bar{x}_1, \bar{x}_2, ..., \bar{x}_{M_l})$, $\bar{y} = (\bar{y}_1, \bar{y}_2, ..., \bar{y}_{M_l})$, $\bar{x}_i, \bar{y}_i \in D_l^{M_l}$ $(1 \leqslant i \leqslant M_l)$.

(3) $MB = \{MB_l \mid l = 1, 2, ...\}$, corresponding to the special case of *MI* in which I is prime and $M = I - 1$. Let $I_l = p_l$, $D_l = Q(I_l)$, $M_l = I_l - 1$, $N_l = M_l^2$, $E_l = Q(M_l \cdot I_l^2)$, $\bar{z}(l)$ as in MA_l. Then $MB_l: D_l^{N_l} \to E_l^{N_l}$ is given by:

$$MB_l(\bar{x}) = \bar{y} = \bar{z} * \bar{x},$$

where $\bar{x} = (\bar{x}_1, \bar{x}_2, ..., \bar{x}_{M_l})$, $\bar{y} = (\bar{y}_1, y_2, ..., \bar{y}_{M_l})$, $\bar{x}_i \in D_l^{M_l}$, $\bar{y}_i \in E_l^{M_l}$ $(1 \leqslant i \leqslant M_l)$.

We now discuss *computation graphs* which were introduced in [3]. Consider a general sequential random-access computer program P, which can compute an

infinite family of functions $\{G_l \mid l = 1, 2, \ldots\}$, where $G_l: D_l^{N_l} \to E_l^{N_l}$, D_l and E_l being finite domains and $|D_l| = R_l$. We denote G_l by $G_l(x_1, x_2, \ldots, x_{N_l}) = (y_1, y_2, \ldots, y_{N_l})$. The x_i's $(1 \leqslant i \leqslant N_l)$ are called *inputs* and the y_i's $(1 \leqslant i \leqslant N_l)$ are called *outputs*.

Now consider any fixed l, and let $R = R_l$, $D = D_l = \{d_1, d_2, \ldots, d_R\}$, $E = E_l = \{e_1, e_2, \ldots, e_H\}$, $N = N_l$, $G = G_l$. All possible computations of the program P on inputs from D^N can be described by a *D^N-computation graph* (or simply, a *computation graph*). This is a rooted directed graph whose nodes represent storage states of the program. (For a Turing machine with random-access input heads, e.g., a storage state includes the contents of the work tapes, the internal state, and the positions of all heads.) Each nonsink u in the graph represents a state in which a certain input x_j is queried and from which a new state, depending on the value of x_j, is reached. Thus each such u has R outgoing edges, labeled $d_1, d_2, \ldots, d_R$. An edge labeled d_q leading from node u to node v means that if at the storage state represented by u, the program finds that the value of x_j is d_q, then the program transfers to the storage state represented by v. Note that we assume that x_j is known if the program accesses any of its bits. This assumption cannot increase the time spent by the program (see also [3]). With each node of the computation graph a list of pairs of the form $y_{i_j} \leftarrow e_{k_j}$ may be associated. If the list $((y_{i_1} \leftarrow e_{k_1}), \ldots, (y_{i_r} \leftarrow e_{k_r}))$ is associated with node u $(1 \leqslant i_j \leqslant N, 1 \leqslant k_j \leqslant H, 1 \leqslant j \leqslant r)$, this means that at the storage state represented by u, the program outputs value e_{k_j} for y_{i_j} $(1 \leqslant j \leqslant r)$. We shall say that the node u *produces* the above outputs.

Given an input vector $(x_1, x_2, \ldots, x_N) = (a_1, a_2, \ldots, a_N) \in D^N$, the computation of the program on this input is traced by following a directed path in the computation graph as follows: The first node on the path is the root. For each node u on the path, the next node v on the path is determined as follows: Let the input queried at u be x_j. Then follow the edge labeled a_j (we say that the answer to the x_j *query* is a_j on this path), and v is the node to which this edge leads. This process continues until a sink is reached. We refer to this path as *the path traced by the input vector* $(x_1, x_2, \ldots, x_N) = (a_1, a_2, \ldots, a_N)$. The output produced by the program for y_i $(1 \leqslant i \leqslant N)$ is b_i if a pair $y_i \leftarrow b_i$ is associated with a node on the above path. (We assume that on the same path from the root we cannot find two output pairs $y_i \leftarrow b$ and $y_i \leftarrow b'$ with $b \neq b'$).

We now consider the time and space used by the program. For the corresponding computation graph, let T denote the maximum, over all input vectors, of the length of the path traced by the input vector. Clearly T is a lower bound on the worst case time $\hat{T}$ spent by the program on inputs from D^N. Now let the *capacity* S of the computation graph be the logarithm of the number of nodes (base 2 is assumed for logarithms unless another base is specified). We claim that up to a constant factor, S is a lower bound on the worst case space used by the program on input vectors from D^N. This is seen as follows (see [3, 12]): Suppose that the storage of the program is represented over a finite alphabet with K symbols. If $\hat{S}$ is the worst case space then the number of possible storage states is at most $K^{\hat{S}}$, hence:

$$2^S \leqslant K^{\hat{S}} \qquad \text{hence} \qquad S \leqslant \log K \cdot \hat{S}.$$

Since a computation graph represents a program which computes a function G as above we make the following definitions.

DEFINITION 2.1. A *D^N-computation graph for a function* $G: D^N \to E^N$, where $|D| = R$, is a rooted directed graph as described above such that for every $(a_1, a_2, ..., a_N) \in D^N$, $G(a_1, a_2, ..., a_N) = (b_1, b_2, ..., b_N)$ if and only if on the path traced by the input vector $(x_1, x_2, ..., x_N) = (a_1, a_2, ..., a_N)$, a pair $y_i \leftarrow b_i$ is encountered for $1 \leqslant i \leqslant N$, and if a pair $y_j \leftarrow c_j$ is encountered, then $c_j = b_j$.

DEFINITION 2.2. A *partial D^N-computation graph for a function* $G: D^N \to E^N$, where $|D| = R$, is a rooted directed graph as described above such that for every $(a_1, a_2, ..., a_N) \in D^N$, if $G(a_1, a_2, ..., a_N) = (b_1, b_2, ..., b_N)$ then on the path traced by the input vector $(x_1, x_2, ..., x_N) = (a_1, a_2, ..., a_N)$ as above, if a pair $y_i \leftarrow c_i$ is encountered then $c_i = b_i$. (The intuitive meaning is that no incorrect outputs are computed but for each input vector only a possibly empty subset of the outputs is computed.)

In both definitions we assume that a path traced by any input vector cannot include the same node twice. Clearly such a possibility implies that on some input vector the corresponding program never halts.

We note that a D^N-computation graph is a special case of a partial D^N-computation graph. For any partial D^N-computation graph τ, the *output produced by* τ for some input vector $(x_1, x_2, ..., x_N) = (a_1, a_2, ..., a_N)$ is the set of all pairs of the form $y_i \leftarrow b_i$ which are associated with the nodes on the path traced by this input vector. If c is a node on this path, we say that the input vector $(a_1, a_2, ..., a_N)$ *leads to* node c. We will sometimes omit D^N and use the terms *computation graph* or *partial computation graph*. Two (partial) D^N-computation graphs τ and τ' are said to be *equivalent* if for every input vector $(a_1, a_2, ..., a_N) \in D^N$ τ produces the same output as τ'.

With an increase of T and S by at most a constant factor, we can transform a (partial) D^N-computation graph τ into an equivalent (partial) D^N-computation graph τ' with the following properties:

(1) It is acyclic.

(2) For every node there is an input vector leading to it.

(3) The set of nodes may be partitioned into levels numbered 1, 2,..., and edges from nodes in level j lead only to nodes in level $j + 1$.

As pointed out in [3], (1) and (3) can be achieved by a construction analogous to a construction due to Pippenger, presented in [12], for comparison branching programs. We now describe the construction for our case: given τ, let τ' have T copies of the node set of τ. If in a node v of τ x_j is queried then all copies of v also query x_j. If an edge leads from a node v to a node u in τ, then in τ' we have a copy of this edge (with its input and output labels) from copy i of v to copy $i + 1$ of u for $1 \leqslant i \leqslant T - 1$. Clearly τ' is equivalent to τ and has the same T and capacity $S' = S + \log T$. Tompa [12] proved that $\log T \leqslant S$, and the argument, modified to

apply to our case, is as follows: T is the length of a path traced by some input vector $\bar{x}_0$. Since no node appears twice on this path, clearly T is at most the number of nodes in τ, which is at most 2^S. Hence indeed $T \leqslant \log S$. Now, in order to satisfy (2) above, delete from τ' any node to which no input vector leads.

From now on we will asume that every (partial) computation graph satisfies the above three properties. This can affect a lower bound on the product $T \cdot S$ only by a constant factor. Also, T can be redefined as the *depth* of the graph, which is the length of the longest directed path in it.

We now formalize the properties of functions which enable us to obtain time–space tradeoff results. In the sequel D will denote a finite domain.

DEFINITION 2.3. A function f defined on D^N is said to *depend on all of its arguments* if for every $1 \leqslant j \leqslant N$ and $(t_1, t_2, ..., t_N) \in D^N$ there exists $(s_1, s_2, ..., s_N) \in D^N$ such that $t_i = s_i$ for $i \neq j$ and $f(t_1, t_2, ..., t_N) \neq f(s_1, s_2, ..., s_N)$.

Notation 2.1. Let $g: D^N \to D^N$ be a function of N variables $x_1, x_2, ..., x_N$. Let

$$U = \{i_1, i_2, ..., i_r\} \subset \{1, 2, ..., N\} \qquad (1 \leqslant i_1 < i_2 < \cdots < i_r \leqslant N).$$

Let $\bar{u} = (u_1, u_2, ..., u_r) \in D^r$. Then $g^{[U,\bar{u}]}$ denotes the function with domain D^{N-r} obtained from g by substituting u_k for x_{i_k} $(1 \leqslant k \leqslant r)$.

Notation 2.2. Let $f: D^N \to D^N$ be a function;

$$f(x_1, x_2, ..., x_N) = (y_1, y_2, ..., y_N),$$

where $y_i = f_i(x_1, x_2, ..., x_N)$ $(1 \leqslant i \leqslant N)$.
Let

$$U = \{i_1, i_2, ..., i_r\} \qquad (1 \leqslant i_1 < i_2 < \cdots < i_r \leqslant N),$$
$$V = \{j_1, j_2, ..., j_s\} \qquad (1 \leqslant j_1 < j_2 < \cdots j_s \leqslant N),$$
$$\bar{u} = (u_1, u_2, ..., u_r) \in D^r, \qquad \bar{v} = (v_1, v_2, ..., v_s) \in D^s.$$

Then $\psi(f, (U, \bar{u}), (V, \bar{v}))$ denotes the system of equations in $N - r$ unknowns:

$$f_{j_k}^{[U,\bar{u}]}(x_{q_1}, x_{q_2}, ..., x_{q_{N-r}}) = v_k \qquad (1 \leqslant k \leqslant s),$$

where

$$\{q_1, q_2, ..., q_{N-r}\} = \{1, 2, ..., N\} - \{i_1, i_2, ..., i_r\}.$$

$\theta(f, (U, \bar{u}), (V, \bar{v}))$ denotes the number of solutions of this system.

Remark. For the sake of simplicity we made Notation 2.1 and 2.2 with x_i, y_i indexed by elements of $\{1, ..., N\}$. Later we will use also double indexing, and x_{ij}, y_{ij}

will be viewed as indexed by elements of, say, $\{1, 2,..., K\} \times \{1, 2,..., M\}$. U, V will then be subsets of $\{1, 2,..., K\} \times \{1, 2,..., M\}$.

DEFINITION 2.4. Let f be a function, $f: D^N \to D^N$, where $|D| = R$. Let m be an integer, $0 < m \leqslant N$, and α a real number, $0 \leqslant \alpha < 1$. Then f is called (m, α)-*mixing* if for any $U \subset \{1, 2,..., N\}$, $V \subset \{1, 2,..., N\}$ such that $r = |U| \leqslant m$, $s = |V| \leqslant m$ and any $\bar{u} \in D^r$, $\bar{v} \in D^s$,

$$\theta(f, (U, \bar{u}), (V, \bar{v})) \leqslant R^{N-r-\alpha s}.$$

Intuitively, the following lemma, essentially due to Borodin and Cook [3], states that if a function f is (m, α)-mixing, then any partial computation graph for f of small depth cannot produce many outputs for a large number of input vectors.

LEMMA 2.1 (Essentially due to Borodin and Cook [3]). *Let $f: D^N \to D^N$ be an (m, α)-mixing function, $|D| = R$, and τ a partial D^N-computation graph for f, of depth r, $r \leqslant m$. Then for every $s \leqslant m$, the number of input vectors in D^N for which τ produces at least s outputs is at most $R^{N-\alpha s}$.*

Proof. By duplicating nodes we can transform τ into a tree of depth r. Then we may extend τ in such a way that along every path from the root exactly r inputs are queried. We may also associate all outputs with the leaves, by associating with each leaf all the outputs produced on the path from the root to it. All these transformations do not change the depth of τ, and result in an equivalent partial computation graph. Now, consider any leaf c, such that s outputs $y_{j_k} \leftarrow v_k$ $(1 \leqslant k \leqslant s)$, where $1 \leqslant j_2 < j_2 < \cdots < j_s \leqslant N$ are associated with c. Let $x_{i_1}, x_{i_2},..., x_{i_r}$, where $1 \leqslant i_1 < i_2 < \cdots < i_r \leqslant N$, be the inputs queried along the path from the root to c, and let u_k be the answer to the x_{i_k} query on this path $(1 \leqslant k \leqslant r)$. Let

$$U = \{i_1, i_2,..., i_r\}, \qquad V = \{j_1, j_2,..., j_s\},$$
$$\bar{u} = (u_1, u_2,..., u_r), \qquad \bar{v} = (v_1, v_2,..., v_s).$$

Now, an input vector leading to c must satisfy the system $\psi(f, (U, \bar{u}), (V, \bar{v}))$. Since f is (m, α)-mixing, there are at most $R^{N-r-\alpha s}$ such input vectors. Since there are at most R^r leaves in a tree of depth r and out-degree R, at most $R^{N-\alpha s}$ input vectors can produce at least s outputs each.

THEOREM 2.1 ("Borodin and Cook's counting principle," essentially due to Borodin and Cook [3]). *There exists a constant a such that: For every $b > 1$ there exists an N_0 such that if D is a finite domain, $|D| = R$, $R^\alpha > b^2$, $N \geqslant N_0$, and $f: D^N \to D^N$ is an (m, α)-mixing function which depends on all of its arguments, then for every computation graph τ for f*

$$T \cdot S \geqslant a \cdot \log b \cdot N \cdot m.$$

Proof. The proof relies on Lemma 2.1. Both Lemma 2.1 and Theorem 2.1 were proved in [3] in a somewhat different form. A condition apparently weaker than (m, α)-mixing was assumed there. For our purposes it is enough to assume that f is (m, α)-mixing.

We start by noticing that $T \geqslant N$, since f depends on all of its arguments. This is seen as follows: Suppose $T < N$, and choose any input vector $(x_1, x_2, \ldots, x_N) = (t_1, t_2, \ldots, t_N)$. For some j, x_j is not queried on the path traced by $(t_1, \ldots, t_N)$. There exists an input vector $(s_1, s_2, \ldots, s_N)$ with $s_i = t_i$ for $i \neq j$ and $f(s_1, s_2, \ldots, s_N) \neq f(t_1, t_2, \ldots, t_N)$.

The path traced by $(s_1, s_2, \ldots, s_N)$ is the same as the path traced by $(t_1, t_2, \ldots, t_N)$. Hence the same output is produced for both input vectors, contradicting $f(t_1, t_2, \ldots, t_N) \neq f(s_1, s_2, \ldots, s_N)$. So indeed $T \geqslant N$, hence $S \geqslant \log N$. Let

$$S_1 = \frac{S}{\log b}, \qquad N_0 \geqslant b.$$

If $N \geqslant N_0$ then $S_1 \geqslant 1$. Now, if $S_1 > m$ then

$$T \cdot S > \log b \cdot N \cdot m.$$

If, however, $S_1 \leqslant m$, divide the computation graph τ into stages, each of depth m. For $1 \leqslant i \leqslant \lfloor N/S_1 \rfloor$, let K_i be the number of distinct input vectors for which the first i stages produce at least $i \cdot S_1$ outputs. Consider a node v at the last level of stage i. By Lemma 2.1, the number of input vectors which lead to v and for which at least S_1 outputs are produced in stage $i+1$, is at most $R^{N-\alpha S_1}$. Hence, since there are at most 2^S such v,

$$K_{i+1} \leqslant K_i + 2^S R^{N-\alpha S_1} \qquad \text{for } i \geqslant 1.$$

Also by Lemma 2.1,

$$K_1 \leqslant R^{N-\alpha S_1} \leqslant 2^S \cdot R^{N-\alpha S_1}.$$

By induction on i,

$$K_i \leqslant 2^S \cdot i \cdot R^{N-\alpha S_1} \qquad \left(1 \leqslant i \leqslant \left\lfloor \frac{N}{S_1} \right\rfloor\right).$$

Letting $i_0 = \lfloor N/S_1 \rfloor$ we get

$$\begin{aligned} K_{i_0} &\leqslant \frac{N}{S_1} \cdot \frac{R^N}{(R^\alpha)^{S_1}} \cdot 2^S = \frac{N}{S_1} \cdot \frac{R^N}{(R^\alpha)^{S_1}} \cdot b^{S_1} \\ &\leqslant \frac{R^N}{S_1} \cdot \left(\frac{b^2}{R^\alpha}\right)^{S_1} \qquad \left(b^{S_1} \geqslant N \text{ since } S_1 \geqslant \frac{\log N}{\log b} = \log_b N\right), \\ &< R^N \qquad \left(\text{since } S_1 \geqslant 1 \text{ and } \left(\frac{b^2}{R^\alpha}\right) < 1\right). \end{aligned}$$

Hence in the first i_0 stages, for some input vector, less than $i_0 \cdot S_1 \leqslant N$ outputs are produced. Hence

$$T \geqslant i_0 \cdot m = \left\lfloor \frac{N}{S_1} \right\rfloor \cdot m.$$

If $S_1 < \frac{1}{2}N$ then

$$\left\lfloor \frac{N}{S_1} \right\rfloor \cdot m \geqslant \left(\frac{N}{S_1} - 1 \right) m = \frac{N - S_1}{S_1} \cdot m \geqslant \frac{N \cdot m}{2S_1},$$

hence

$$T \cdot S_1 \geqslant \tfrac{1}{2}N \cdot m.$$

If $S_1 \geqslant \frac{1}{2}N$ then

$$T \cdot S_1 \geqslant N \cdot \tfrac{1}{2}N \geqslant \tfrac{1}{2}N \cdot m.$$

Hence in any case

$$T \cdot S \geqslant \tfrac{1}{2} \log b \cdot N \cdot m. \quad \blacksquare$$

3. Time–Space Tradeoffs and a Proof of Theorem 1.1

We see from Theorem 2.1 that the (m, α)-mixing property for a function f implies a lower bound on $T \cdot S$ for any computation graph for f. The following Lemma 3.1 is useful in showing that functions are (m, α)-mixing.

Lemma 3.1 (The combination principle). *Let D be a finite domain, $R = |D|$. Let*

$$\bar{x} = (x_{i1}, x_{i2}, \ldots, x_{iM}), \qquad \bar{y}_i = (y_{i1}, y_{i2}, \ldots, y_{iM}) \ (1 \leqslant i \leqslant K),$$

and let g be a function of the variables x_{ij} $(1 \leqslant i \leqslant K, 1 \leqslant j \leqslant M)$,

$$g: D^N \to D^N \qquad \text{where} \quad N = KM$$

and

$$g(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_K) = (\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_K),$$

where $\bar{y}_i = g_i(\bar{x}_i)$, $g_i: D^M \to D^M$ $(1 \leqslant i \leqslant K)$, and suppose that all the g_i are (m, α)-mixing. Then g is also (m, α)-mixing.

Proof. Let $\bar{K} = \{1, 2, \ldots, K\}$, $\bar{M} = \{1, 2, \ldots, M\}$,

$$U, V \subset \bar{K} \times \bar{M}, \qquad r = |U| \leqslant m, \qquad s = |V| \leqslant m, \qquad \bar{u} \in D^r, \qquad \bar{v} \in D^s,$$
$$W_i = \{i\} \times \bar{M}, \qquad V_i = V \cap W_i, \qquad U_i = U \cap W_i \ (1 \leqslant i \leqslant K).$$

Let Z be the set of solutions of the system $\psi(g, (U, \bar{u}), (V, \bar{v}))$. Z is the set-theoretic direct product of sets Z_i, where Z_i is the set of solutions of a system

$$\psi(g_i, (U_i, \bar{u}_i), (V_i, \bar{v}_i)) \qquad (1 \leqslant i \leqslant K),$$

where

$$|U_i| = r_i, \qquad |V_i| = s_i, \qquad \bar{u}_i \in D^{r_i}, \qquad \bar{v}_i \in D^{s_i},$$

$$\sum_{i=1}^{K} r_i = r, \qquad \sum_{i=1}^{K} s_i = s.$$

Since for $1 \leqslant i \leqslant K$ g_i is (m, α)-mixing and

$$r_i \leqslant r \leqslant m, \qquad s_i \leqslant s \leqslant m,$$

$$|Z_i| \leqslant R^{M-\alpha s_i}$$

hence $|Z| = \prod_{i=1}^{K} |Z_i| \leqslant \prod_{i=1}^{K} R^{M-\alpha s_i} = R^{KM-\alpha s} = R^{N-\alpha s}$. ∎

Intuitively, the following simple lemma states that for any discrete Fourier transform matrix over a finite field, all submatrices with a small enough number of rows and large enough number of columns have rank which is at least half of the number of rows.

LEMMA 3.2. *Let D be a finite field, w a primitive Nth root of unity in D, and A an $N \times N$ matrix whose (i, j) entry is $w^{(i-1)(j-1)}$. Then every submatrix B of A having s rows and $N - r$ columns with $r, s \leqslant \frac{1}{4}N$ has rank at least $\frac{1}{2}s$. Hence, the function $f: D^N \to D^N$ given by $f(\bar{x}) = A \cdot \bar{x}$ is $(\frac{1}{4}N, \frac{1}{2})$-mixing.*

Proof. By a result due to Tompa [12, 13], any submatrix of A consisting of s rows and s *consecutive* columns has rank s. Let B_1 be the submatrix of A consisting of the s rows of B and all the N columns. The number of B_1's columns which are also in B is at least $\frac{3}{4}N$. Divide the columns of B_1 into consecutive blocks, each having s consecutive columns, except for, maybe, the last one which has between 1 and s columns. Since $s \leqslant \frac{1}{4}N$ and $N - r \geqslant \frac{3}{4}N$, the above blocks, excluding the last one, contain together at least $\frac{1}{2}N$ columns of B. Hence at least one block of s consecutive columns has in it at least $\frac{1}{2}s$ columns of B. By Tompa's result all these $\frac{1}{2}s$ columns of B are linearly independent. ∎

We now use Theorem 2.1 and Lemmas 3.1, 3.2 in order to obtain time–space tradeoff results. We say that a property of a parameter l holds for *almost every* l if for some l_0 it holds for $l \geqslant l_0$.

LEMMA 3.3. *There exists a constant k_1 such that for almost every l, if τ_l is a computation graph for F_l (a function related to the discrete Fourier transform, see Sect. 2), then for τ_l*

$$T \cdot S \geqslant k_1 \cdot N_l^2.$$

Proof. By Lemma 3.2, F_l is $(\frac{1}{4}N_l, \frac{1}{2})$-mixing. Clearly, F_l depends on all of its arguments. The result follows by Theorem 2.1. ∎

LEMMA 3.4. *There exists a constant k_2 such that for almost every l, if τ_l is a computation graph for MA_l(a function related to matrix multiplication over finite prime fields, see Sect. 2), then for τ_l*

$$T \cdot S \geqslant k_2 \cdot M_l^3.$$

Proof. If we choose the same sequence of primitive roots of unity w_l in the definition of both F_l and MA_l we get

$$MA_l(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_{M_l}) = (F_l(\bar{x}_1), F_l(\bar{x}_2), \ldots, F_l(\bar{x}_{M_l})).$$

By Lemma 3.2, F_l is $(\frac{1}{4}M_l, \frac{1}{2})$-mixing, hence by Lemma 3.1, MA_l is also $(\frac{1}{4}M_l, \frac{1}{2})$-mixing. Clearly MA_l depends on all of its arguments. The result follows by Theorem 2.1. ∎

LEMMA 3.5. *There exists a constant k_3 such that for almost every l, if τ_l is a computation graph for MB_l (a function related to matrix multiplication over the integers, see Sect. 2), then for τ_l*

$$T \cdot S \geqslant k_3 \cdot M_l^3.$$

Proof. From τ_l we can construct τ_l', a computation graph for MA_l, by replacing all input and output values by their value modulo p_l, and deleting redundant edges. Clearly T and S are not increased by this transformation, and the result follows by Lemma 3.4 ∎

Lemmas 3.3, 3.4, 3.5 are sufficient for obtaining lower bounds on the product *time · space* for the discrete Fourier transform over finite prime fields (DFT), for matrix multiplication over finite prime fields (MF), and for matrix multiplication over the integers (MI), for infinitely many N. For MF and MI, however, we can obtain lower bounds for almost every M, using a "padding argument" and the following result from number theory:

Bertrand's Postulate (see [9]). For every integer $q \geqslant 1$ there exists a prime p, $q < p \leqslant 2q$.

Now, for every natural number $l \geqslant 3$, choose a prime q_l, $\lfloor l/2 \rfloor < q_l \leqslant l$. Define the following infinite families of functions:

(1) $MA1$ (corresponding to MF): Let $M_l = l - 1$, $D_l = GF(q_l)$, $N_l = M_l^2$, w_l a primitive $(q_l - 1)$th root of unity modulo q_l, $\bar{z}(l)$ an M_l by M_l matrix over D_l given by:

$$\bar{z}(l) = (\bar{z}(l)_1, \bar{z}(l)_2, \ldots, \bar{z}(l)_{M_l})^T,$$

where

$$z(l)_{ij} = w^{(i-1)(j-1)} \pmod{q_l} \qquad \text{if } 1 \leqslant i, j \leqslant q_l - 1,$$
$$z(l)_{ij} = 0 \qquad \text{if } i > q_l - 1 \text{ or } j > q_l - 1.$$

Then $MA1_l: D_l^{N_l} \to D_l^{N_l}$ is given by

$$MA1_l(\bar{x}) = \bar{z} * \bar{x} \pmod{q_l}.$$

(2) $MB1$ (corresponding to MI): Let $M_l = l - 1$, $D_l = Q(q_l)$, $N_l = M_l^2$, $E_l = Q(q_l^2 \cdot M_l)$, w_l a primitive $(q_l - 1)$th root of unity modulo q_l, $\bar{z}(l)$ an M_l by M_l matrix over the integers given by:

$$\bar{z}(l) = (\bar{z}(l)_1, \bar{z}(l)_2, \ldots, \bar{z}(l)_{M_l}),$$

where

$$z(l)_{ij} = w_l^{(i-1)(j-1)} \pmod{q_l} \qquad \text{if } 1 \leqslant i, j \leqslant q_l - 1,$$
$$z(l)_{ij} = 0 \qquad \text{if } i > q_l - 1 \text{ or } j > q_l - 1.$$

Then $MB1_l: D_l^{N_l} \to E_l^{N_l}$ is given by

$$MB1_l(\bar{x}) = \bar{z} * \bar{x}.$$

LEMMA 3.6. *There exists a constant k_4 such that for every l, if τ_l is a computation graph for $MA1_l$ or for $MB1_l$ then for τ_l*

$$T \cdot S \geqslant k_4 \cdot M_l^3.$$

Proof. First consider $MA1_l$. Let t be such that p_t (the tth prime) is equal to q_l. Then from τ_l we can construct a computation graph τ_t' for MA_t by deleting redundant nodes and edges. Let S', T' be the capacity and depth, respectively, of τ_t'. Let S, T be the corresponding values for τ_l. Let M_t be as in the definition of MA_t, and let M_l be as in the definition of $MA1_l$. We have

$$T' \leqslant T \qquad S' \leqslant S,$$

and

$$M_l = l - 1 \leqslant 2(q_l + 1) = 2(p_t + 1) = 2(M_t + 2).$$

From Lemma 3.4. we know that for t large enough

$$T' \cdot S' \geqslant k_2 \cdot M_t^3.$$

Hence we get for l large enough

$$T \cdot S \geqslant \frac{k_2}{8}(M_l - 4)^3.$$

Now clearly for l large enough, $M_l - 4 \geqslant \frac{1}{2}M_l$, and the result for $MA1$ follows. The result for $MB1$ follows in a similar way from Lemma 3.5. ∎

Our main result, namely Theorem 1.1, follows from Lemmas 3.3, 3.6, and the fact that every program for the discrete Fourier transform can compute the infinite family of functions F, every program for matrix multiplication over finite prime fields can compute the infinite family of functions $MA1$, and every program for matrix multiplication over the integers can compute the infinite family of functions $MB1$. As we have mentioned before, T is a lower bound on *time* and S is a lower bound on *space*.

As we have mentioned before, the above lower bounds are optimal up to a factor of $\log n^{O(1)}$ for the above-mentioned range of entries length. We think that this range is of particular interest. For instance, Professor Stephen A. Cook (personal communication) has drawn the author's attention to the fact that if the entries length is large, say $\Omega(2^N)$, then the complexity of matrix multiplication over the integers is closely related to the complexity of multiplying two integers.

4. Concluding Remarks

We have obtained new results concerning the complexity of classical algebraic computational problems on unrestricted models of sequential computation. While nontrivial time bounds are still lacking for those problems on such models, we can prove nontrivial time bounds when space is restricted.

Regarding the techniques, we find it quite interesting that properties of the discrete Fourier transform matrix which were used by Tompa [12] to prove similar time–space tradeoffs on the restricted model of an arithmetic straight-line program turned out to be useful in proving time–space tradeoffs on unrestricted models. A similar phenomenon is mentioned in [3]: The proof, appearing in [4], for sorting on a restricted model was helpful in constructing a proof for the unrestricted model.

In this paper we use "Borodin and Cook's counting principle" in a systematic, unified way, by introducing the concept of (m, α)-mixing functions. We also show that the (m, α)-mixing property is preserved under a certain operation. We also use transformations on families of computation graphs, which can be viewed as nonuniform reducibilities among problems.

Acknowledgment

The author would like to thank Professor Stephen A. Cook for encouragement and helpful discussions.

REFERENCES

1. A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, "The Design and Analysis of Computer Algorithms," Addison–Wesley, Reading, Mass., 1974.
2. A. BORODIN, "Structured vs. General Models in Computational Complexity," University of Toronto, Department of Computer Science Technical Report 143/80, July 1980.
3. A. BORODIN AND S. COOK, A time–space tradeoff for sorting on a general sequential model of computation, *SIAM J. Comput.* **11** (2) (1982), 287–297.
4. A. BORODIN, M. J. FISCHER, D. KIRKPATRICK, N. LYNCH, AND M. TOMPA, A time–space tradeoff for sorting on non-oblibious machines, *in* "Proceedings IEEE 20th Annual Found. of Comput. Sci. Conference (1979)," pp. 319–327.
5. A. COBHAM, The recognition problem for the set of perfect squares, *in* "Conference Record IEEE 1966 Seventh Annual Symposium on Switching and Automata Theory," pp. 78–87.
6. P. DURIS AND Z. GALIL, A time–space tradeoff for language recognition, *in* "Proceedings IEEE 22nd Annual Found. of Comput. Sci. Conference (1981)," pp. 53–57.
7. D. YU GRIGORYEV, An application of separability and independence notions for proving lower bounds for circuit complexity, *in* "Notes on Scientific Seminars 60," pp. 38–48, Steklov Mathematical Institute, Leningrad, 1976. [Russian]
8. J. HOPCROFT AND J. ULLMAN, "Introduction to Automata Theory, Languages, and Computation," Addison–Wesley, Reading, Mass., 1979.
9. G. HARDY AND E. WRIGHT, "An Introduction to the Theory of Numbers," pp. 343, 373, Oxford Univ. Press, London/New York, 1965.
10. J. JA'JA', Time–space tradeoffs for some algebraic problems, *in* "Proceedings 12th Annual ACM Sympos. Theory Comput., (1980)," pp. 339–349.
11. S. REISCH AND G. SCHNITGER, Three applications of Kolmogorov-complexity, *in* "Proceedings IEEE 23rd Annual Found. of Comput. Sci. Conference (1982)," pp. 45–52.
12. M. TOMPA, "Time–Space Tradeoffs for Straight-Line and Branching Programs," University of Toronto, Department of Computer Science Technical Report 122/78, 1978.
13. M. TOMPA, Time–space tradeoffs for computing functions, using connectivity properties of their circuits, *in* "Proceedings 10th Annual Sympos. Theory Comput., (1978)," 196–204.